

# Underapproximative verification of stochastic LTI systems using Fourier transform and convex optimization

This example will demonstrate the use of SReach in verification and controller synthesis for stochastic continuous-state discrete-time linear time-invariant (LTI) systems.

Specifically, we will discuss the [terminal hitting-time stochastic reach-avoid problem](#), where we are provided with a stochastic system model, a safe set to stay within, and a target set to reach at a specified time, and we will use SReach to solve the following problems:

1. **Verification problem from an initial state:** Compute an [underapproximation of the maximum attainable reach-avoid probability given an initial state](#),
2. **Controller synthesis problem:** Synthesize a [controller to achieve this probability](#), and
3. **Verification problem:** Compute a [polytopic underapproximation](#) of all the initial states from which the system can be driven to meet a predefined probabilistic safety threshold.

Our approach uses Fourier transforms, convex optimization, and gradient-free optimization techniques to compute a scalable underapproximation to the [terminal hitting-time stochastic reach-avoid problem](#).

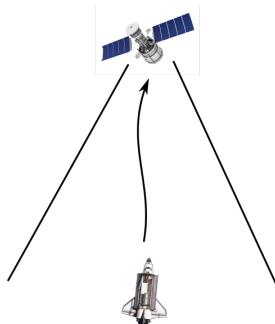
## Notes about this Live Script:

1. **MATLAB dependencies:** This Live Script uses MATLAB's [Global Optimization Toolbox](#), and [Statistics and Machine Learning Toolbox](#).
2. **External dependencies:** This Live Script uses Multi-Parametric Toolbox (MPT) and CVX.
3. We will also [Genz's algorithm](#) (included in helperFunctions of SReach) to evaluate integrals of a Gaussian density over a polytope.
4. Make sure that `sreachinit` is run before running this script.

This Live Script is part of the SReach toolbox. License for the use of this function is given in <https://github.com/abyvinod/SReach/blob/master/LICENSE>.

## Problem formulation: spacecraft rendezvous and docking problem

We consider both the spacecrafts, referred to as the deputy spacecraft and the chief spacecraft, to be in the same circular orbit. **We desire that the deputy reaches the chief at a specified time (the control time horizon) while remaining in a line-of-sight cone.** To account for the modeling uncertainties and unmodeled disturbance forces, we will use a stochastic model to describe the relative dynamics of the deputy satellite with respect to the chief satellite.



## Dynamics model for the deputy relative to the chief spacecraft

The relative planar dynamics of the deputy with respect to the chief are described by the [Clohessy-Wiltshire-Hill \(CWH\) equations](#). Specifically, we have a LTI system describing the relative dynamics and it is perturbed by a low-stochasticity Gaussian disturbance to account for unmodelled phenomena and disturbance forces. We will set the thrust levels permitted to be within a origin-centered box of side 0.2.

```
umax=0.1;
mean_disturbance = zeros(4,1);
covariance_disturbance = diag([1e-4, 1e-4, 5e-8, 5e-8]);
% Define the CWH (planar) dynamics of the deputy spacecraft relative to the chief spacecraft a
sys = getCwhLtiSystem(4,...
    Polyhedron('lb', -umax*ones(2,1),...
               'ub',  umax*ones(2,1)),...
    StochasticDisturbance('Gaussian',...
                           mean_disturbance,...
                           covariance_disturbance));
```

## Target set and safe set creation

For the formulation of the [terminal hitting-time stochastic reach-avoid problem](#),

- **the safe set** is the line-of-sight (LoS) cone is the region where accurate sensing of the deputy is possible (set to avoid is outside of this LoS cone), and
- **the target set** is a small box around the origin which needs to be reached (the chief is at the origin in the relative frame).

```
time_horizon=5; % Stay within a line of sight cone
                % reach the target at t=5% Safe Set
%% Safe set definition --- LoS cone  $|x| \leq y$  and  $y \in [0, y_{\max}]$  and  $|v_x| \leq v_{x\max}$  and  $|v_y| \leq v_{y\max}$ 
ymax=2;
vxmax=0.5;
vymax=0.5;
A_safe_set = [1, 1, 0, 0;
              -1, 1, 0, 0;
               0, -1, 0, 0;
               0, 0, 1, 0;
               0, 0, -1, 0;
               0, 0, 0, 1;
               0, 0, 0, -1];
b_safe_set = [0;
              0;
              ymax;
              vxmax;
              vxmax;
              vymax;
              vymax];
safe_set = Polyhedron(A_safe_set, b_safe_set);
%% Target set --- Box  $[-0.1, 0.1] \times [-0.1, 0] \times [-0.01, 0.01] \times [-0.01, 0.01]$ 
target_set = Polyhedron('lb', [-0.1; -0.1; -0.01; -0.01],...
                        'ub', [0.1; 0; 0.01; 0.01]);
```

## Problem 1 and 2: Verification and controller synthesis from a given initial state

We will first specify the initial state and parameters for the MATLAB's Global Optimization Toolbox `patternsearch`.

```

initial_state = [-0.75;          % Initial x relative position
                 -0.75;          % Initial y relative position
                 0;              % Initial x relative velocity
                 0];             % Initial y relative velocity
slice_at_vx_vy = initial_state(3:4);
%% Parameters for MATLAB's Global Optimization Toolbox patternsearch
desired_accuracy = 1e-3;        % Decrease for a more accurate lower
                                % bound at the cost of higher
                                % computation time
PSoptions = psoptimset('Display','off');

```

Next, using `SReach`, we will compute an [optimal open-controller and the associated reach-avoid probability](#). This function takes about few minutes to run.

```

[lb_stochastic_reach_avoid, optimal_input_vector] = ...
    getFtBasedLowerBoundOnStochasticReachAvoidProblem(sys,...
                                                       initial_state,...
                                                       time_horizon,...
                                                       safe_set,...
                                                       target_set,...
                                                       [],...
                                                       desired_accuracy,...
                                                       PSoptions);

```

The function `getFtLowerBoundStochasticReachAvoid` uses [Fourier transform and convex optimization](#) to underapproximate the reach-avoid problem. Note that

`lb_stochastic_reach_avoid` is a lower bound to the maximum attainable reach-avoid probability since using a state-feedback law (also known as a Markov policy) can incorporate more information and attain a higher threshold of safety. Unfortunately, the current state-of-the-art approaches can compute a state-feedback law only using [dynamic programming](#) (intractable for a 4D problem) or provide [overapproximations](#) of safety (unsuitable for verification).

Using the computed optimal open-loop control law, we can compute the associated optimal mean trajectory.

```

[H_matrix, mean_X_sans_input, ~] =...
    getHmatMeanCovForXSansInput(sys,...
                                initial_state,...
                                time_horizon);
optimal_mean_X = mean_X_sans_input + H_matrix * optimal_input_vector;
optimal_mean_trajectory = reshape(optimal_mean_X, sys.state_dimension, []);

```

## Visualization of the optimal mean trajectory and the safe and target sets

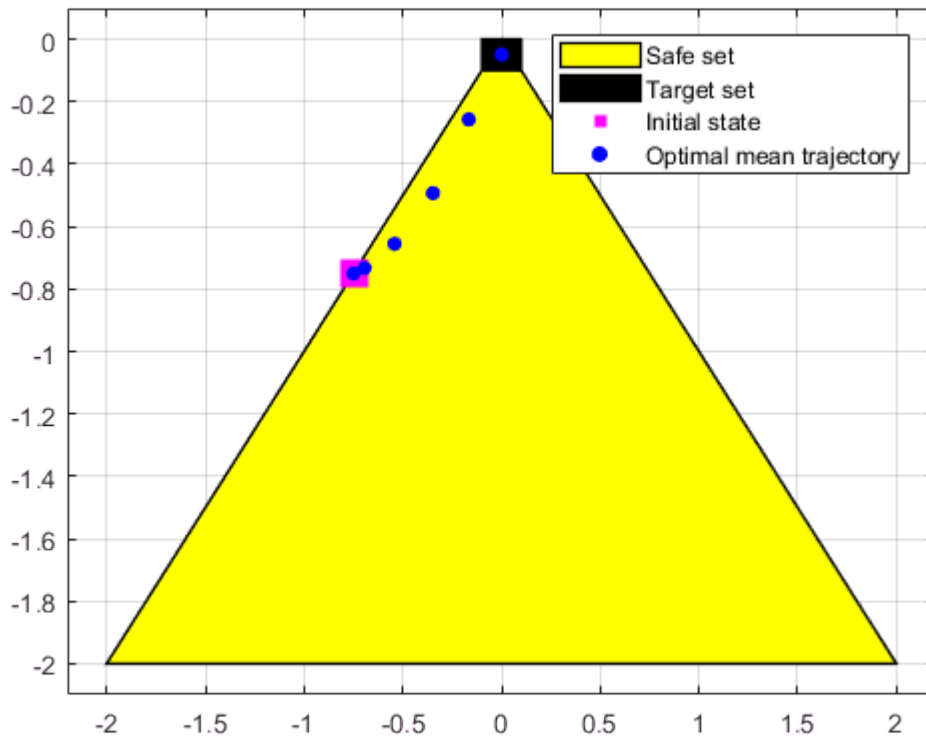
We can visualize this trajectory along with the specified safe and target sets using MPT3's plot commands.

```

%% Plotting
figure();
box on;
hold on;
plot(safe_set.slice([3,4], slice_at_vx_vy), 'color', 'y');
plot(target_set.slice([3,4], slice_at_vx_vy), 'color', 'k');
scatter(initial_state(1), initial_state(2), 200, 'ms', 'filled');
scatter([initial_state(1), optimal_mean_trajectory(1,:)], ...
        [initial_state(2), optimal_mean_trajectory(2,:)], ...

```

```
30, 'bo', 'filled');
legend_cell = {'Safe set', 'Target set', 'Initial state', 'Optimal mean trajectory'};
legend(legend_cell);
```



## Validate the open-loop controller and the obtained lower bound using Monte-Carlo simulations

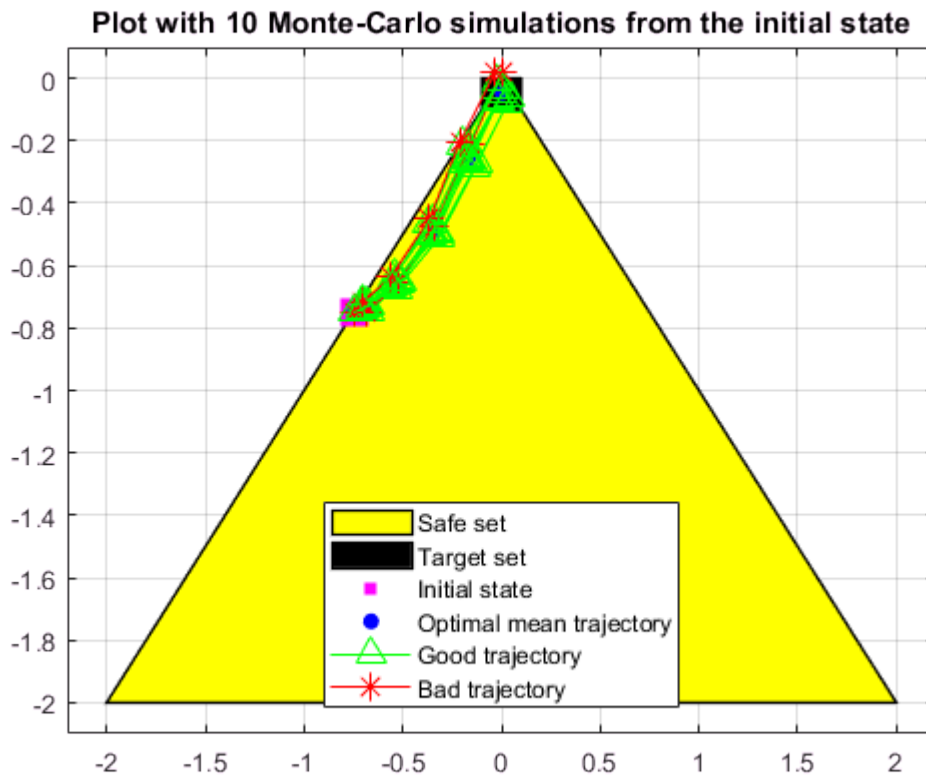
```
figure();
box on;
hold on;
plot(safe_set.slice([3,4], slice_at_vx_vy), 'color', 'y');
plot(target_set.slice([3,4], slice_at_vx_vy), 'color', 'k');
scatter(initial_state(1), initial_state(2), 200, 'ms', 'filled');
scatter([initial_state(1), optimal_mean_trajectory(1,:)], ...
        [initial_state(2), optimal_mean_trajectory(2,:)], ...
        30, 'bo', 'filled');
legend_cell = {'Safe set', 'Target set', 'Initial state', 'Optimal mean trajectory'};
legend(legend_cell);
%% Monte-Carlo simulation parameters
no_mcarlo_sims = 100000;
no_sims_to_plot = 10;

[reach_avoid_probability_mcarlo, ...
 legend_cell] = checkViaMonteCarloSims(no_mcarlo_sims, ...
                                       sys, ...
                                       initial_state, ...
                                       time_horizon, ...
                                       safe_set, ...
                                       target_set, ...
                                       optimal_input_vector, ...
                                       legend_cell);
```

```

no_sims_to_plot);
leg = legend(legend_cell, 'Location', 'South');
title(sprintf('Plot with %d Monte-Carlo simulations from the initial state',...
no_sims_to_plot));
box on;
grid on;

```



```

fprintf(['Open-loop-based lower bound and Monte-Carlo simulation ',...
'(%1.0e particles): %1.3f, %1.3f\n'],...
no_mcarlo_sims,...
lb_stochastic_reach_avoid,...
round(reach_avoid_probability_mcarlo / desired_accuracy) *...
desired_accuracy);

```

Open-loop-based lower bound and Monte-Carlo simulation (1e+05 particles): 0.865, 0.860

### Problem 3: Computation of an underapproximative stochastic reach-avoid set

We will now compute a [polytopic underapproximation](#) using the convexity and compactness properties of these sets. Specifically, we can compute the projection of the stochastic reach-avoid set on a 2-dimensional hyperplane on the set of all initial states.

For this example, we consider a hyperplane that fixes the initial velocity. This example sets an initial velocity of  $[0.1 \ 0.1]^T$ . We also specify other parameters needed for this approach. We will reuse the `LtiSystem` object as well as the safe sets and the target sets, `safe_set` and `target_set`.

```

% Definition of the affine hull
slice_at_vx_vy = ones(2,1)*0.01;
affine_hull_of_interest_2D_A = [zeros(2) eye(2)];
affine_hull_of_interest_2D_b = slice_at_vx_vy;
% The initial velocities of interest

```

```

affine_hull_of_interest_2D = Polyhedron('He',...
                                         [affine_hull_of_interest_2D_A,...
                                          affine_hull_of_interest_2D_b]);

%% Other parameters of the problem
time_horizon=5;
probability_threshold_of_interest = 0.8;    % Stochastic reach-avoid 'level' of interest
no_of_direction_vectors = 8;               % Increase for a tighter polytopic
                                           % representation at the cost of higher
                                           % computation time

tolerance_bisection = 1e-2;                % Tolerance for bisection to compute the
                                           % extension

%% Parameters for MATLAB's Global Optimization Toolbox patternsearch
desired_accuracy = 1e-3;                   % Decrease for a more accurate lower
                                           % bound at the cost of higher
                                           % computation time

PSoptions = psoptimset('Display','off');

```

Construct the polytopic underapproximation of the stochastic reach-avoid set. The function `getFtBasedUnderapproximateStochasticReachAvoidSet` will provide the polytope ( $n$ -dimensional) and the optimal open-loop controllers for each of the vertices, along with other useful information. This function will take  $\sim 20$  minutes to run.

```

[underapproximate_stochastic_reach_avoid_polytope,...
 optimal_input_vector_at_boundary_points,...
 xmax,...
 optimal_input_vector_for_xmax,...
 maximum_underapproximate_reach_avoid_probability,...
 optimal_theta_i,...
 optimal_reachAvoid_i] =...
    getFtBasedUnderapproximateStochasticReachAvoidSet(...
                                                sys,...
                                                time_horizon,...
                                                safe_set,...
                                                target_set,...
                                                probability_threshold_of_interest,...
                                                tolerance_bisection,...
                                                no_of_direction_vectors,...
                                                affine_hull_of_interest_2D,...
                                                desired_accuracy,...
                                                PSoptions);

```

Computing the  $x_{\max}$  for the Fourier transform-based underapproximation  
Polytopic underapproximation exists for  $\alpha = 0.80$  since  $W(x_{\max}) = 0.865$ .

Analyzing direction (shown transposed) :1/8  
-1      0      0      0

Upper bound of theta: 0.41

OptRAProb	OptTheta	LB_theta	UB_theta	OptInp^2	Exit reason
0.8660	0.2059	0.0000	0.4117	0.0099	Feasible
0.8660	0.3088	0.2059	0.4117	0.0114	Feasible
0.8610	0.3603	0.3088	0.4117	0.0197	Feasible
0.8600	0.3860	0.3603	0.4117	0.0201	Feasible
0.8610	0.3988	0.3860	0.4117	0.0245	Feasible
0.8580	0.4053	0.3988	0.4117	0.0265	Feasible

Analyzing direction (shown transposed) :2/8  
-0.7071    -0.7071      0      0

Upper bound of theta: 1.39

OptRAProb	OptTheta	LB_theta	UB_theta	OptInp^2	Exit reason
0.8650	0.0000	0.0000	1.3933	0.0095	Infeasible

0.8650	0.0000	0.0000	0.6966	0.0095	Infeasible
0.8660	0.1742	0.0000	0.3483	0.0223	Feasible
0.8660	0.2612	0.1742	0.3483	0.0318	Feasible
0.8660	0.3048	0.2612	0.3483	0.0303	Feasible
0.8660	0.3265	0.3048	0.3483	0.0273	Feasible
0.8660	0.3374	0.3265	0.3483	0.0258	Feasible
0.8660	0.3429	0.3374	0.3483	0.0255	Feasible

Analyzing direction (shown transposed) :3/8  
-0.0000   -1.0000   0   0

Upper bound of theta: 0.99

OptRAProb	OptTheta	LB_theta	UB_theta	OptInp^2	Exit reason
0.8650	0.0000	0.0000	0.9852	0.0095	Infeasible
0.8660	0.2463	0.0000	0.4926	0.0101	Feasible
0.8660	0.3694	0.2463	0.4926	0.0173	Feasible
0.8660	0.4310	0.3694	0.4926	0.0211	Feasible
0.8660	0.4618	0.4310	0.4926	0.0236	Feasible
0.8660	0.4772	0.4618	0.4926	0.0248	Feasible
0.8650	0.4849	0.4772	0.4926	0.0250	Feasible

Analyzing direction (shown transposed) :4/8  
0.7071   -0.7071   0   0

Upper bound of theta: 1.39

OptRAProb	OptTheta	LB_theta	UB_theta	OptInp^2	Exit reason
0.8650	0.0000	0.0000	1.3933	0.0095	Infeasible
0.8650	0.0000	0.0000	0.6966	0.0095	Infeasible
0.8660	0.1742	0.0000	0.3483	0.0163	Feasible
0.8660	0.2612	0.1742	0.3483	0.0211	Feasible
0.8660	0.3048	0.2612	0.3483	0.0269	Feasible
0.8660	0.3265	0.3048	0.3483	0.0363	Feasible
0.8660	0.3374	0.3265	0.3483	0.0352	Feasible
0.8660	0.3429	0.3374	0.3483	0.0339	Feasible

Analyzing direction (shown transposed) :5/8  
1.0000   -0.0000   0   0

Upper bound of theta: 1.62

OptRAProb	OptTheta	LB_theta	UB_theta	OptInp^2	Exit reason
0.8650	0.0000	0.0000	1.6179	0.0095	Infeasible
0.8650	0.0000	0.0000	0.8089	0.0095	Infeasible
0.8660	0.2022	0.0000	0.4045	0.0154	Feasible
0.8660	0.3034	0.2022	0.4045	0.0187	Feasible
0.8660	0.3539	0.3034	0.4045	0.0273	Feasible
0.8660	0.3792	0.3539	0.4045	0.0266	Feasible
0.8660	0.3918	0.3792	0.4045	0.0214	Feasible
0.8660	0.3982	0.3918	0.4045	0.0256	Feasible

Analyzing direction (shown transposed) :6/8  
0.7071   0.7071   0   0

Upper bound of theta: 1.14

OptRAProb	OptTheta	LB_theta	UB_theta	OptInp^2	Exit reason
0.8650	0.0000	0.0000	1.1440	0.0095	Infeasible
0.8650	0.0000	0.0000	0.5720	0.0095	Infeasible
0.8660	0.1430	0.0000	0.2860	0.0134	Feasible
0.8660	0.2145	0.1430	0.2860	0.0120	Feasible
0.8660	0.2503	0.2145	0.2860	0.0140	Feasible
0.8660	0.2681	0.2503	0.2860	0.0156	Feasible
0.8660	0.2771	0.2681	0.2860	0.0177	Feasible

Analyzing direction (shown transposed) :7/8  
0.0000   1.0000   0   0

Upper bound of theta: 0.41

OptRAProb	OptTheta	LB_theta	UB_theta	OptInp^2	Exit reason
0.8630	0.2059	0.0000	0.4117	0.0143	Feasible
0.8660	0.3088	0.2059	0.4117	0.0232	Feasible
0.8660	0.3603	0.3088	0.4117	0.0304	Feasible
0.8640	0.3860	0.3603	0.4117	0.0307	Feasible
0.8630	0.3988	0.3860	0.4117	0.0316	Feasible
0.8550	0.4053	0.3988	0.4117	0.0328	Feasible

Analyzing direction (shown transposed) :8/8  
-0.7071    0.7071    0    0

Upper bound of theta: 0.29

OptRAProb	OptTheta	LB_theta	UB_theta	OptInp^2	Exit reason
0.8640	0.1456	0.0000	0.2911	0.0168	Feasible
0.8660	0.2183	0.1456	0.2911	0.0286	Feasible
0.8660	0.2547	0.2183	0.2911	0.0290	Feasible
0.8650	0.2729	0.2547	0.2911	0.0302	Feasible
0.8340	0.2820	0.2729	0.2911	0.0301	Feasible

## Visualization of the underapproximative polytope and the safe and target sets

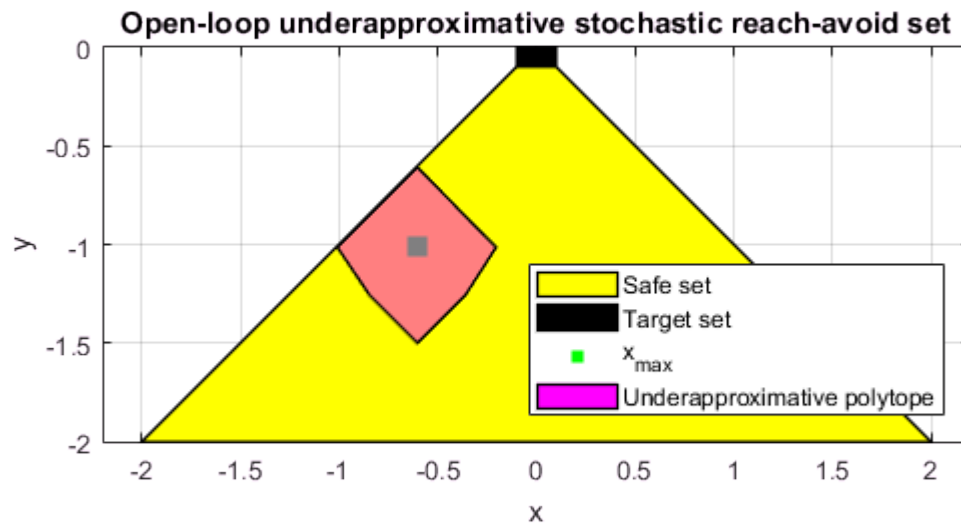
Construct the 2D representation of the underapproximative polytope.

```
set_of_direction_vectors = computeSetOfDirectionVectors(...  
                                                                no_of_direction_vectors,...  
                                                                sys.state_dimension,...  
                                                                affine_hull_of_interest_2D);  
vertex_poly = xmax + optimal_theta_i.* set_of_direction_vectors;  
underapproximate_stochastic_reach_avoid_polytope_2D =...  
                                                                Polyhedron('V',vertex_poly(1:2,:));
```

Plot the underapproximative polytope along with the safe and the target sets.

```
figure();  
hold on;  
plot(safe_set.slice([3,4], slice_at_vx_vy), 'color', 'y');  
plot(target_set.slice([3,4], slice_at_vx_vy), 'color', 'k');  
  
scatter(xmax(1), xmax(2), 100, 'gs', 'filled')  
if ~isEmptySet(underapproximate_stochastic_reach_avoid_polytope)  
    plot(underapproximate_stochastic_reach_avoid_polytope_2D,...  
         'color','m','alpha',0.5);  
    leg=legend({'Safe set',...  
               'Target set',...  
               'x_{max}',...  
               'Underapproximative polytope'});  
else  
    leg=legend({'Safe set', 'Target set', 'x_{max}'})  
end  
set(leg, 'Location', 'SouthEast');  
xlabel('x')  
ylabel('y')  
axis equal  
box on;  
grid on;  
title('Open-loop underapproximative stochastic reach-avoid set');
```





## Validate the underapproximative set and the controllers synthesized using Monte-Carlo simulations

```

if ~isEmptySet(underapproximate_stochastic_reach_avoid_polytope)
    for direction_index = 1:no_of_direction_vectors
        figure();
        hold on;
        plot(safe_set.slice([3,4], slice_at_vx_vy), 'color', 'y');
        plot(target_set.slice([3,4], slice_at_vx_vy), 'color', 'k');
        scatter(vertex_poly(1,direction_index),...
                vertex_poly(2,direction_index),...
                200,'cs','filled');
        plot(underapproximate_stochastic_reach_avoid_polytope_2D,...
            'color','m','alpha',0.5);
        legend_cell = {'Safe set',...
                       'Target set',...
                       'Initial state',...
                       'Underapproximation set'};

        [reach_avoid_probability_mcarlo,...
         legend_cell] = checkViaMonteCarloSims(...
            no_mcarlo_sims,...
            sys,...
            vertex_poly(:,direction_index),...
            time_horizon,...
            safe_set,...
            target_set,...
            optimal_input_vector_at_boundary_points(:, direction_index),...
            legend_cell,...
            no_sims_to_plot);
        % Compute and plot the mean trajectory under the optimal open-loop

```

```

% controller from the the vertex under study
[H_matrix, mean_X_sans_input, ~] =...
    getHmatMeanCovForXSansInput(sys,...
        vertex_poly(:,direction_index),...
        time_horizon);
optimal_mean_X = mean_X_sans_input + H_matrix *...
    optimal_input_vector_at_boundary_points(:, direction_index);
optimal_mean_trajectory=reshape(optimal_mean_X,sys.state_dimension,[]);
% Plot the optimal mean trajectory from the vertex under study
scatter(...
    [vertex_poly(1,direction_index), optimal_mean_trajectory(1,:)],...
    [vertex_poly(2,direction_index), optimal_mean_trajectory(2,:)],...
    30, 'bo', 'filled');
legend_cell{end+1} = 'Mean trajectory';
leg = legend(legend_cell, 'Location', 'EastOutside');
% title for the plot
if no_sims_to_plot > 0
    title(sprintf(['Open-loop-based lower bound: %1.3f\n Monte-Carlo ',...
        'simulation (%d out of %1.0e plotted): ',...
        '%1.3f\n'],...
        optimal_reachAvoid_i(direction_index),...
        no_sims_to_plot,...
        no_mcarlo_sims,...
        round(reach_avoid_probability_mcarlo / desired_accuracy) *...
        desired_accuracy));
end
box on;
grid on;

fprintf(['Open-loop-based lower bound and Monte-Carlo simulation ',...
    '(%1.0e particles): %1.3f, %1.3f\n'],...
    no_mcarlo_sims,...
    optimal_reachAvoid_i(direction_index),...
    round(reach_avoid_probability_mcarlo / desired_accuracy) *...
    desired_accuracy);
end
end

```

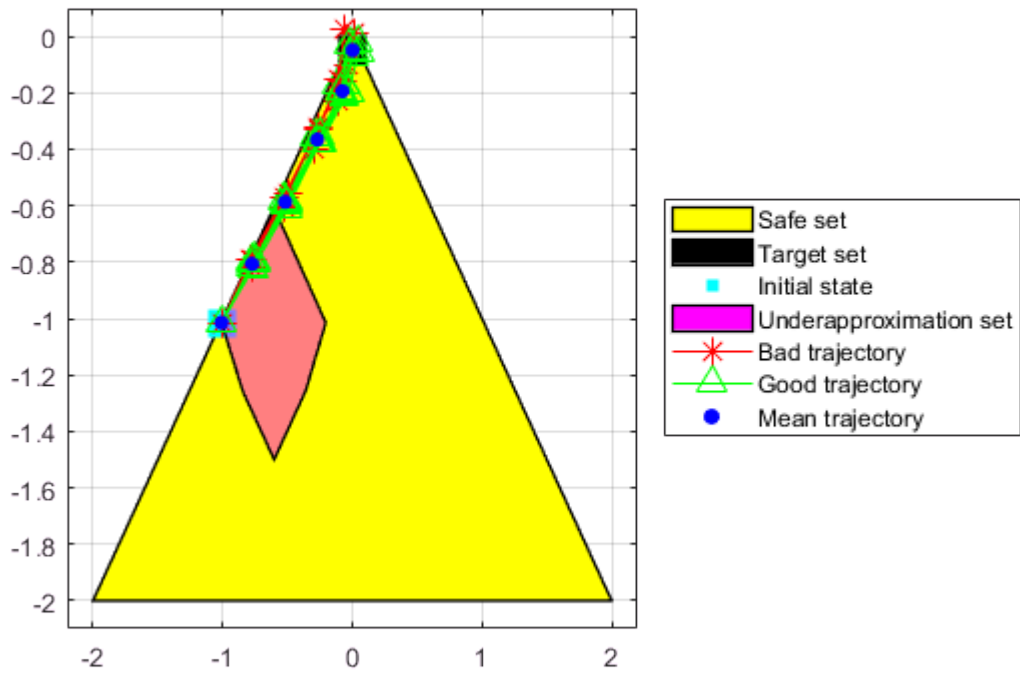
```

Open-loop-based lower bound and Monte-Carlo simulation (1e+05 particles): 0.858, 0.858
Open-loop-based lower bound and Monte-Carlo simulation (1e+05 particles): 0.866, 0.866
Open-loop-based lower bound and Monte-Carlo simulation (1e+05 particles): 0.865, 0.862
Open-loop-based lower bound and Monte-Carlo simulation (1e+05 particles): 0.866, 0.869
Open-loop-based lower bound and Monte-Carlo simulation (1e+05 particles): 0.866, 0.868
Open-loop-based lower bound and Monte-Carlo simulation (1e+05 particles): 0.866, 0.864
Open-loop-based lower bound and Monte-Carlo simulation (1e+05 particles): 0.855, 0.855
Open-loop-based lower bound and Monte-Carlo simulation (1e+05 particles): 0.834, 0.831

```

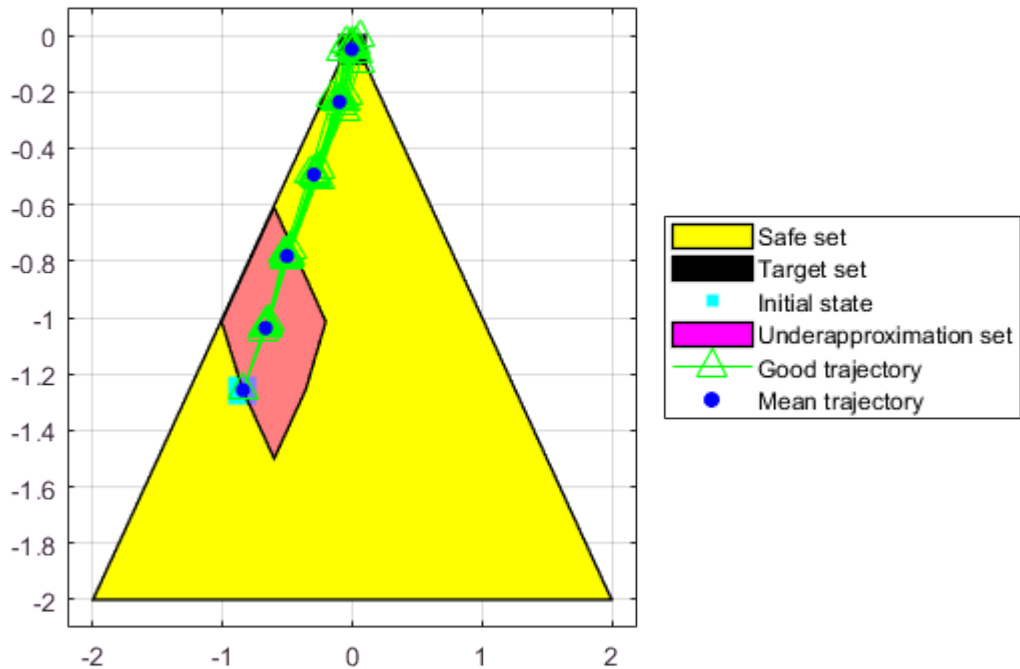
Open-loop-based lower bound: 0.858

Monte-Carlo simulation (10 out of 1e+05 plotted): 0.858



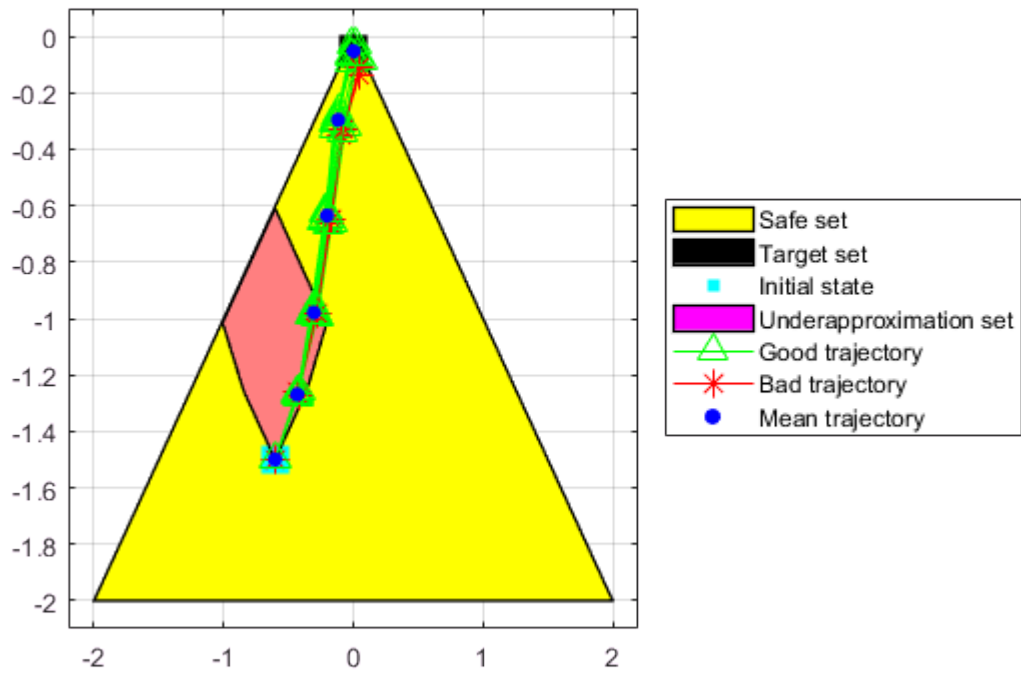
Open-loop-based lower bound: 0.866

Monte-Carlo simulation (10 out of 1e+05 plotted): 0.866



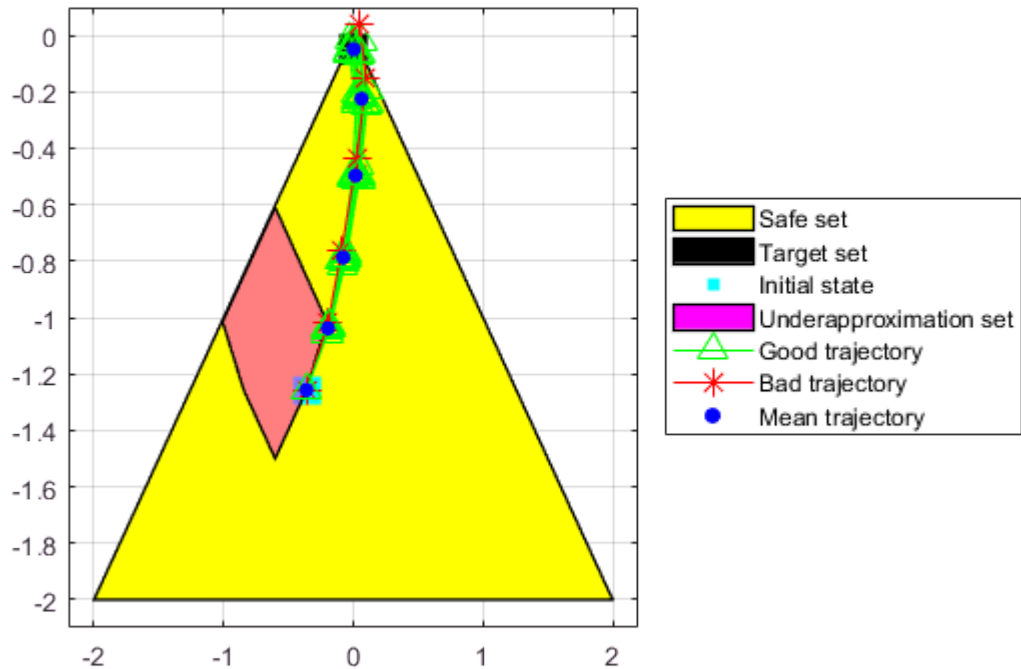
Open-loop-based lower bound: 0.865

Monte-Carlo simulation (10 out of 1e+05 plotted): 0.862



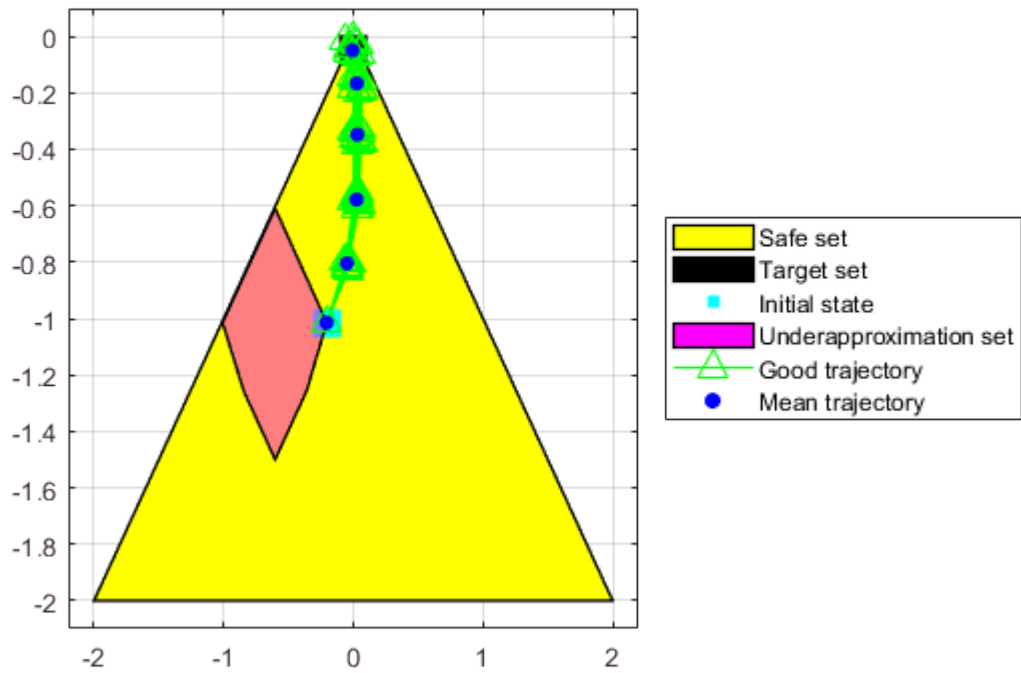
Open-loop-based lower bound: 0.866

Monte-Carlo simulation (10 out of 1e+05 plotted): 0.869



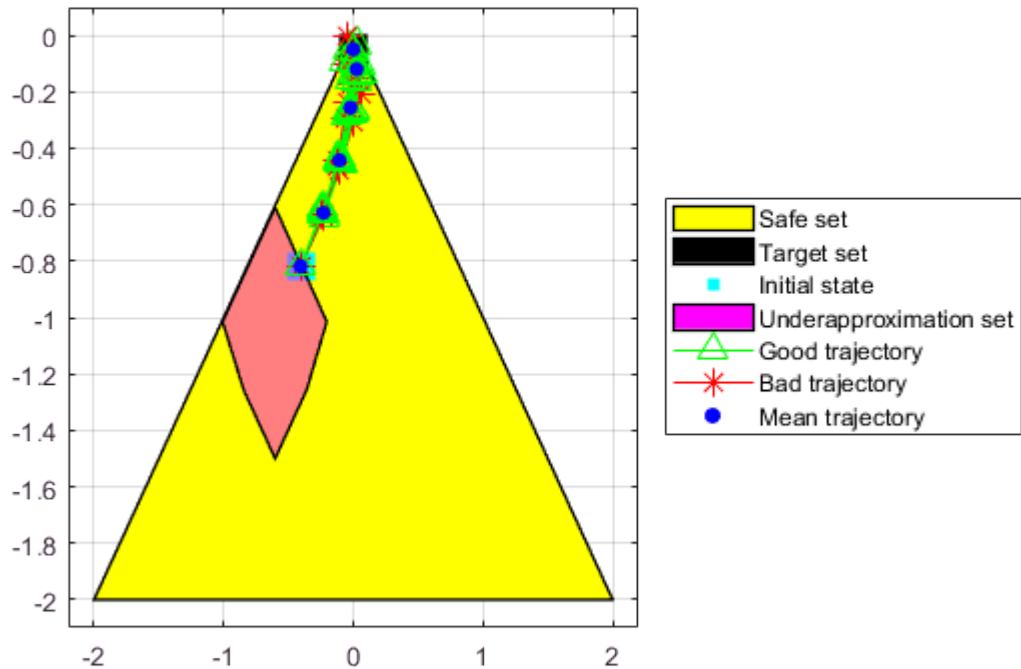
Open-loop-based lower bound: 0.866

Monte-Carlo simulation (10 out of 1e+05 plotted): 0.868



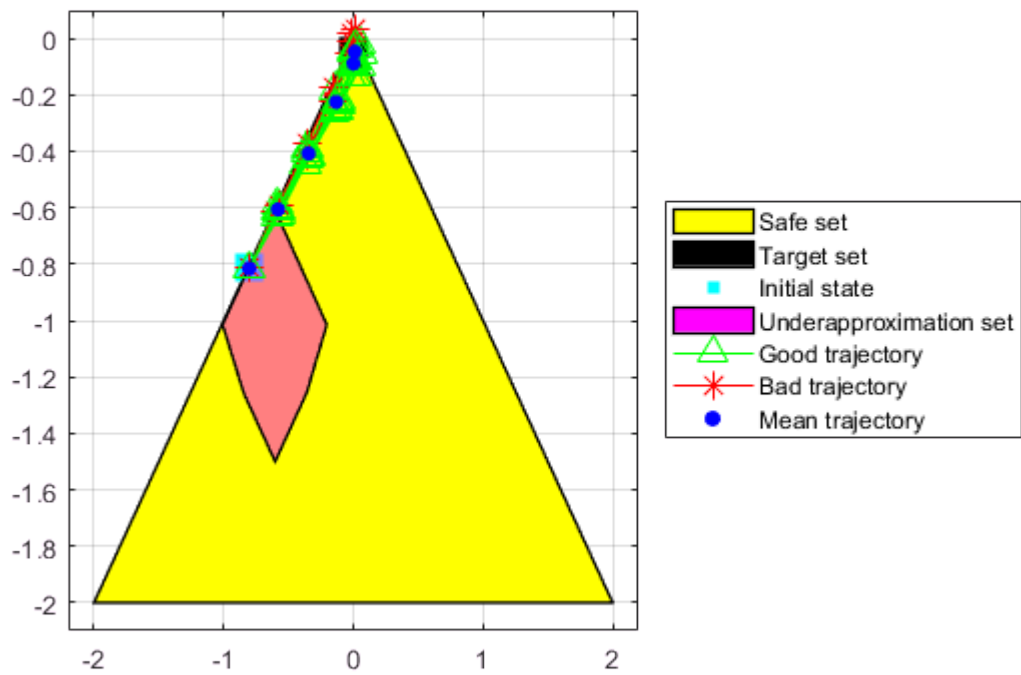
Open-loop-based lower bound: 0.866

Monte-Carlo simulation (10 out of 1e+05 plotted): 0.864



Open-loop-based lower bound: 0.834

Monte-Carlo simulation (10 out of 1e+05 plotted): 0.831



Open-loop-based lower bound: 0.855

Monte-Carlo simulation (10 out of 1e+05 plotted): 0.855

